JOURNAL OF INNOVATION IN DIGITAL ECOSYSTEMS 1 (2014) 26–37

HOSTED BY

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/jides

A typed applicative system for a language and text processing engineering

*Ismaïl Biskri**, *Marie Anastacio*, *Adam Joly*, *Boucif Amar Bensaber*

LAMIA – DMI, Université du Québec à Trois-Rivières, CP 500, Trois-Rivières, Québec, Canada

ARTICLE INFO

Article history:
Received 23 October 2014
Received in revised form
19 January 2015
Accepted 15 February 2015
Published online 4 March 2015

Keywords:
Processing chains
Language
Text analysis
Categorial and applicative systems
Combinatory logic

ABSTRACT

In this paper, we present a flexible, modular, consistent, and coherent approach for language and text processing engineering. Each processing chain dedicated to text processing is regarded as a serial or parallel assembly of modules, underlying particular tasks a user wants to apply to a text. Users, according to their needs and perspectives might want to build and validate their own processing chain by assembling a set of modules according to a certain configuration. In this paper, we suggest a theoretical formal system based on the model of the typed applicative grammars and the combinatory logic. This approach allows providing a general framework in which users would be able to build multiple language and text analysis processes according to their own objectives. It will also systematize the verification of the logical consistency of the sequence of modules in the assembly that characterizes a given processing chain.

1. Introduction

Nowadays, what is known as 'Language and Text processing' is all the fields related to information retrieval, categorization, classification, indexation, syntactic analysis, semantic analysis, knowledge extraction, knowledge management, etc. These fields are fundamental; they can impact economical, scientific, political, cultural and social sectors. This is particularly amplified by the fact that corpora, textual databases, and mainly the web (especially social networks), represent an endless source of information.

Recently, some voices have been raised among the scientific community to denounce the actual limits of these fields. The critics object based on the following hypothesis; the domain expert – or alternatively the computer scientist expert – would be the designer of an implemented system that would only require periodical updates.

This hypothesis has been proven to be unproductive, given the fact that it does not take into account the subjectivity, or the point of view of the user, may he be an expert or not on the topic.

Furthermore, the hypothesis stated above does not allow the "collaboration of multiple points of view", that often originates from different disciplines such as computer science, artificial intelligence, linguistic, psychology, semiology, logic, philosophy, terminology, ontology, etc.; reading and analysis

Peer review under responsibility of Qassim University.

* *Corresponding author*.
E-mail addresses: Ismail.Biskri@uqtr.ca (I. Biskri), Marie.Anastacio@uqtr.ca (M. Anastacio), Adam.Joly@uqtr.ca (A. Joly), Boucif.Amar.Bensaber@uqtr.ca (B. Amar Bensaber).

http://dx.doi.org/10.1016/j.jides.2015.02.003

of texts are considered to be at the intersection of these disciplines.

For example, users who wish to retrieve information with a search engine, like Google, may decide to use one or more downstream filters to refine their results. They may also decide that the combination of the search engine and the filter is the ideal processing chain for their needs, and thus want to keep it for reuse. Therefore, users can create, thereby, a processing chain that meets more specifically their needs. In order to illustrate this concept, consider the case of a "librarian" who wants to retrieve papers of one given author, and access to definitions and to citations contained in these papers. Giving the name of the author as a keyword is not sufficient. Adding the word "definition" and "citation" as keywords will not adjust the result of the query, since those key words are not significant terms in the papers. A linguistic filter as the one presented in [3,4] used downstream of the search engine will allow the extraction of definitions or citations. Librarians can create a processing chain for their specific needs. They can save or reuse this same processing chain, to access to definitions and citations contained in papers from another author.

In fact, it seems that the solution to this type of problem does not simply reside in supplying one or many software tools. Although developed technologies were successful since they were made more available to users, dissatisfaction has been observed among them due to the following significant limitations; (i) they offer a limited set of closed functionalities; (ii) they are often designed within an architecture that has limited communication capabilities with external softwares that would provide additional functionalities; (iii) it is difficult, if not impossible, to integrate new functionalities to the tool without having to rebuild a significant part or all of it; (iv) the sought collaboration between experts and their intermediaries, all while taking into account the copyrights of each of the creators, is also very complicated.

A methodological reflection on the topic is essential, which brings us to what we could consider a new postulate for text processing. In fact, text reading and analysis – the foundations of any function underlying a task in text processing – is a "dynamic" process that allows multiple "point of views" that can lead to different "understandings" and thus, must be undertaken, while taking into consideration "multiple objectives".

We find in the literature, in data-mining and text-mining, projects on the creation of complex processing chains that offer assembling of many functions and operations and the creation of software platforms for language engineering which integrate statistical analysis, such as RapidMiner [19], WEKA [25], D2K/T2K [14] and Knime [23], or linguistic analysis, such as Context [9] and Gate [10]. Although some of these platforms have enabled the collaboration of researchers in projects like NORA and TAPoR, limitations persist, especially for the assembly of modules, which requires knowledge about the platform and in some cases on the programming code. These new platforms highlight the importance of methodological development surrounding the creation of processing chains [24,18].

In our paper, we will present a flexible, modular, consistent, and coherent architecture for text processing, in which each task will be addressed by an autonomous function that is independent from the other functions of the architecture. A formal theoretical framework based on the model of the typed applicative systems and the combinatory logic will be suggested.

Before presenting the formal model itself, we will introduce in the two following sections combinatory logic and the *Applicative and Combinatory Categorial Grammar*.

2. Combinatory logic

The origins of combinatory logic bring us back to the works of Schönfinkel who defined the concept of combinators in 1924 and sometime later, those of Curry and Feys [11]. This notion was introduced with the purpose to bring a logical solution to some paradoxes, such as Russell's Paradox, but also to eliminate the need for variables in mathematics in order to avoid variables telescoping.

Combinators are abstract operators that use others of the same kind to build more complex ones. They act as functions over arguments, within an operator–operands structure. Each specific action is represented by a unique rule called β-reduction rule, which defines the equivalence between a logical expression with a combinatory, versus one with no combinator.

Although many more combinators exist, we demonstrate in this paper that the combinators we used in our works and their corresponding β-reduction rule [11,15] (for other combinators, the reader may refer to [11,13,15]).

Combinator	Role	β-Reduction rule
B	Composition	**B** x y z → x (y z)
C	Permutation	**C** x z y → x y z
S	Distributive composition	**S** x y u → x u (y u)
C*	Type raising	**C***x y → y x
W	Duplication	**W**x y → x y y
$\mathbf{B}^2$	Composition–Power 2	$\mathbf{B}^2$ x y z u → x (y z u)
$\mathbf{C}_2$	Permutation–Distance 2	$\mathbf{C}_2$ x y z u v → x y v z u

B, **C**, **S**, **W** are elementary combinators. The composition combinator **B** combines two typed operators x and y together in order to form the complex typed operator **B** x y that acts on a typed operand z according to the β-reduction rule. The permutation combinator **C** uses a typed operator x in order to build the complex typed operator **C** x such as if x acts on the typed operands y and z, **C** x will act on those typed operands in the reverse order, that is to say z and y. Given the two typed operators x and y, and the typed operand u, the general composition combinator **S** distributes the typed operand u with the two precedent typed operators x and y. (y u) becomes the typed operand of the complex typed operator (x u). The combinator **C*** is applied on a typed operand x (x functions as the operand of y). It allows to build the complex typed operator (**C*** x) in order to apply it to y. Finally, given the binary

operators x, and the operand y, the combinator **W** duplicates y so that the operator x will have two identical arguments.

We can also combine recursively many elementary combinators together, to form an infinitely range of complex combinators. For example, we could have combinatory expressions such as "**B C** x y z u" or "**S B C** x y z u v". Its global action is determined by the successive application of its elementary combinators, from left to right. If we have the combinatory expression "**B B C** x y z u v", the reduction order would be **B**, **B**, and then **C**.

B B C x y z u v
B(**C** x) y z u v
(**C** x) (y z) u v
x u (y z) v

x u (y z) v is called the normal form of **B B C** x y z u v. According to Church–Rosser theorem, the normal form of each combinatory expression, if it exists, is unique. This assumption has one main corollary: two combinatory expressions are equivalent if and only if they reduce to the same normal form (if it exists).

Two specific cases of complex combinators exist: "power combinators" and "distance combinators". In the first case, a power value of *n* reiterates *n* times the action of the combinator χ, such as $\chi^1 = \chi$ and $\chi^n = \mathbf{B}\chi\chi^{n-1}$. Thereby, the action of the expression "$\mathbf{B}^2$ a b c d e" would be given by the following reduction:

$\mathbf{B}^2$ a b c d e
B B B a b c d e
B (**B** a) b c d e
B a (b c) d e
a (b c) d e

In the latter case, an index value of *n* postpones the action of a combinator χ of *n* steps, such as $\chi_0 = \chi$ and $\chi_n = \mathbf{B}^{n-1}\chi$. If we consider the combinatory expression $\mathbf{C}_2$ a b c d e, the action of the complex combinator would be given by the following reduction:

$\mathbf{C}_2$ a b c d e
B C a b c d e
C (a b) c d e
a b d c e

3. Applicative and combinatory categorial grammar

The model chosen in our work is based on applicative and combinatory categorical grammar (ACCG) [7], a model we widely used in natural language processing. ACCG has won its spurs in syntax and functional semantic.

All Categorial Grammar models are founded on explicit logical rules, substituting a purely surface linguistic analysis for an inferential logical calculation. Relying more on the notion of surface structure, it leads to a logical form in order to express meaning. This model has the advantage of being able to represent the intricacies of phrasal units, by way of the operation of the application of an operator to its operand, a universal representation itself. Somewhat forgotten since Husserl (the concepts of categorems and syncategorems), Lesniewski (semantic categories), Adjukiewics, Bar-Hillel and Lambek (Lambek's calculus), the 1970s, 1980s and 1990s witnessed a significant increase of works and research in the domain of Categorial Grammars. The "collective" can be dubbed "Flexible Categorial Grammars", represented by Montague's model of Universal Grammar for a categorial syntax and denotational semantics, by Steedman's Combinatory Categorial Grammar, associating a categorial syntactic analysis and a construction of functional semantics interpretation by way of lambda-calculus, by Harris' operator–operand grammar, by the Applicative Combinatory Categorial Grammar with the addition of metarules to direct the rules of type-raising and composition [7], as well as other generalizations from Lambek's calculus. Among the most recent developments, we find a multimodal version of Combinatory Categorial Grammars [2] introducing modalities and restrictions on the operability of categorial rules in order to eliminate cases of ambiguity, or even the Abstract Categorial Grammar model [12] to describe syntax and semantics.

Putting aside the differences between these approaches and applications, there are three things that stand out in particular in all these models: (i) their use of logical and mathematical methods to account for language, especially syntax; (ii) their distinction of several logical levels of representation of languages including at the very least a linear structure of the observable level and an operator–operand structure of the construction level [21]; (iii) their flexibility and adaptability to several languages. In keeping with French, English [7,22], Dutch [22] and German with LEXGRAM, etc., new languages are also becoming influenced by a trend of Categorial Grammars. The most recent work includes exploratory analyses for non-Indo-European languages, such as relative constructions in Turkish [8], complement forms in –te in Japanese [17], Korean [16] and nominal phrases in Arabic [1].

ACCG suggests a dichotomous perspective on linguistic units. Some of these linguistic units work as operators and others as operands. This aspect is translated by an assignment of categories to the linguistic units in a way to reflect their applicative nature. Categories are orientated types developed from basic types and from two constructive operators '/' and '\'.

(i) N^* (nominal syntagm) and S (sentence) are basic types.

(ii) If X and Y are orientated types then X/Y and X\Y are orientated types. X/Y and X\Y are functional orientated types. A linguistic unit 'u' with the type X/Y (respectively X\Y) is considered to be an operator (or function) whose typed operand Y is positioned on the right (respectively on the left) of operator.

A linguistic unit u with orientated type X will be designed by [X : u].

For instance, the assigned categories to each linguist unit in the sentence *Freedom reinforces democracy* will be as follows:

Freedom: N^*

Reinforces: $(S\backslash N^*)/N^*$

Democracy: N^*

Freedom and *democracy* are the operands of the verb *reinforces*. *Reinforces* is considered as an operator whose first operand in the sentence (*democracy*) is positioned on its right, and the second operand in the sentence (*freedom*) on its left.

ACCG is an extension of Steedman's Combinatory Categorial Grammar (CCG) by a canonical association between combinatory categorial rules and Curry's combinators. Instead of a pure linguistic analysis, ACCG applies, to the categories assigned to linguistic units, an inferential calculus for the simplification of the categorial types. Here are the rules that enable this calculus [7].

Application rules :	[X/Y : u1] + [Y : u2] ----------> [X : (u1 u2)]	[Y : u1] + [X\Y : u2] ----------< [X : (u2 u1)]
Type-raising rules :	[X : u] ----------**>T** [Y/(Y\X) : (**C*** u)]	[X : u] ----------**<T** [Y\(Y/X) : (**C*** u)]
	[X : u] ----------**>Tx** [Y/(Y/X) : (**C*** u)]	[X : u] ----------**<Tx** [Y\(Y\X) : (**C*** u)]
Functional composition rules :	[X/Y : u1] + [Y/Z : u2] ----------**>B** [X/Z : (**B** u1 u2)]	[Y\Z : u1] + [X\Y : u2] ----------**<B** [X\Z : (**B** u2 u1)]
	[X/Y : u1] + [Y\Z : u2] ----------**>Bx** [X\Z : (**B** u1 u2)]	[Y/Z : u1] + [X\Y : u2] ----------**<Bx** [X/Z : (**B** u2 u1)]
Distributive composition rules :	[(X/Y)/Z : u1] + [Y/Z : u2] ----------**>S** [X/Z : (**S** u1 u2)]	[Y\Z : u1] + [(X\Y)\Z : u2] ----------**<S** [X\Z : (**S** u2 u1)]
	[X/Y)\Z : u1] + [Y\Z : u2] ----------**>Sx** [X\Z : (**S** u1 u2)]	[Y/Z : u1] + [(X\Y)/Z : u2] ----------**<Sx** [X/Z : (**S** u2 u1)]

The premises in each rule are concatenations of linguistic units with orientated types considered to be operators or operands, the consequence of each rule is an applicative typed expression with an eventual introduction of one combinator. The composition of two concatenated units introduces the combinator **B**; the type-raising of a unit u introduces the combinator **C***; the distributive composition of two c concatenated units introduces the combinator S.

According to Steedman [22]:

(i) Rules (>), (<), (>**B**), (<**B**), (>**T**), (<**T**), (>**S**) and (<**S**) are theorems of Lambek Calculus which is context-free [20].

(ii) It is likely that adding rules (>**Bx**), (<**Bx**), (>**Tx**), (<**Tx**), (>**Sx**) and (<**Sx**) will induce greater expressive power than context-free grammar.

(iii) The set of the combinatory categorical rules is characterized by three principles:

(1) The principle of Adjacency: Combinatory categorical rules may only apply to adjacent linguistic units.

(2) The principle of Consistency: Combinatory categorical rules must be consistent with how the principal operator should apply to its operand. For instance, the principle of consistency excludes the following rule in which u1 is the principal operator whose operand, according to the type X\Y, must be positioned on its left.

[X\Y : u1] + [Y : u2]

[X : (u1 u2)]

(3) The principle of Inheritance: if the category that results from the application of a combinatory rule is a functional category, then the constructive operator '/' or '\' in that category will be the same as the one in the one defining the position of the corresponding argument in the premise. For instance, the principle of inheritance excludes the following rule in which the constructed operator (**B** u1 u2) applies on its left to an operand of type Z whereas this operand is positioned on the right of u2.

[X/Y : u1] + [Y/Z : u2]

[X\Z : (**B** u1 u2)]

Consider the analysis of the following sentence *Democracy promotes freedom*:

1.	[N* : *democracy*] + [(S\N*)/N* : *promotes*] + [N* : *freedom*]	
2.	[S/(S\N*):(**C*** *democracy*)] + [(S\N*)/N*: *promotes*] + [N*: *freedom*]	(>**T**)
3.	[S/N* : (**B** (**C*** *democracy*) *promotes*)] + [N* : *freedom*]	(>**B**)
4.	[S : ((**B** (**C*** *democracy*) *promotes*) *freedom*)]	(>)
5.	((**B** (**C*** *democracy*) *promotes*) *freedom*)	
6.	((**C*** *democracy*) (*promotes freedom*))	**B**
7.	((*promotes freedom*) *democracy*)	**C***
8.	*promotes freedom democracy*	

The first step consists in assigning categorial types to the lexical units. Those are entries of a dictionary, where each unit is associated to one or more categories.

Steps 2–4 consist in operating the rules of the ACCG in the way to verify the syntactic correctness and build progressively the predicative structures by the introduction of combinators with the syntactic process. Thus, step 2 consists in applying the rule (>**T**) to the linguistic unit: *democracy*. This step introduces the combinator **C***. Applied to the operand *democracy*, **C*** makes it possible to build an operator (**C*** *democracy*) that we compose at step 3 with the operator promotes with using the rule (>**B**). The result is a more complex operator (**B** (**C*** *democracy*) *promotes*). This last operator is applied in step 4 to the operand *freedom*. Obtaining the type S at step 4 guarantees the syntactic correctness of the sentence. Steps 5–8 are a natural deduction, which consists in eliminating the combinators according to the β-reduction rules shown previously. The predicative structure obtained at step 8, *promotes freedom democracy*, represents the functional semantic interpretation of the given sentence: *Democracy promotes freedom*.

A full processing based upon Applicative and Combinatory Categorial Grammar is carried out in two main steps:

(i) The first step is illustrated by the verification of the proper syntactic connection and the construction of predicative structures with some combinators introduced in certain positions of a syntagmatic structure.

(ii) The second step consists in using the β-reduction rules of combinators in order to create a predicative structure that is underlying the observable expression. The obtained expression is an applicative one. ACCG generates processes that associate one applicative structure to one concatenated. What remains to be eliminated are the combinators of obtained expression in order to construct the "normal form" (in the technical meaning of β-reduction) that expresses the functional semantic interpretation.

Therefore, this process takes the form of a compilation.

With such a model, we have analyzed in previous works many complex constructions in French and Arabic like coordination, subordination, sentences with backward modifiers, etc.

4. The formal model for processing chains

As shown in [5,6] our model refers to programs as modules and concerns systems for which the modules are processed in series only, that is, the so-called processing chains. We are particularly interested in language and text processing systems, for which it could be very useful to simply have

Fig. 1 – A representation of a module with one input.

Fig. 2 – A representation of a module with two inputs.

to modify a module for another one with compatible inputs and outputs. A module acts like a mathematic function that takes arguments, processes one specific action and gives a result. Each module is independent and can be seen like a black box: we are only interested by the general function it accomplished and not how it is programmed internally. The modules must also have the capacity to communicate together with the help of a protocol.

A processing chain is a layout of modules. It is governed by three mains rules: (i) the chain must contain at least one module; (ii) the chain must be syntactically correct; (iii) the semantic aspects of the chain are the responsibility of the user to assume that the chosen modules serve the goals of the processing chain. From a formal point of view, a processing chain is an integrated sequence of computational modules dedicated to specific processings, put together in a (pertinent) order according to a process goal determined by the user. A processing chain will have to allow the composition of the modules.

Therefore, it is essential to answer these two fundamental questions:

(i) Given a set of modules, what are the allowable arrangements which lead to coherent processing chains (the syntactic correctness)?
(ii) Given a coherent processing chain, how can we automate (as much as possible) its assessment (in the sense of its calculability).

In order to do so, a formal system is needed. Such a system is at the center of our theoretical model.

Concretely, a module accomplishes an operation which applies to one or many objectal entities from a given type and returns other objectal entities from another type. Here we have the same applicative principle underlying categorial grammars. This principle is, certainly, more natural for computational modules. We can, therefore, assign to each module an applicative type to reflect how it acts on its operands. Note that, applicative types are not orientated, because operands of computational modules are always positioned on its left.

Applicative types are developed by basic types and from one constructive operator "F" as follows:

(i) Basic types are types.

(ii) If x and y are types then Fxy is a type.

We note a module (Fig. 1) as follows: [M1: Fxy] in which M1 is the identifier of the module and Fxy is the type of M1. M1 is then considered to be a function, whose operand is of type x and the result of the application of M1 on X is of type y.

We note the module M2 (Fig. 2) by [M2: Fx_1Fx_2y]. M2 is a function with two operands: X1 and X2. M2 applies on X1 in order to construct a new function (M2 X1) whose operand is X2. The application of (M2 X1) on X2 gives Y. That is the meaning of the type Fx_1Fx_2y.

Within this approach, the processing chains become applicative "combinations" of typed functions. This view is natural for computational modules, given the fact that they are functions (in its general meaning), from the set of inputs to the set of outputs. Such combinations will be interpreted, like in some works in metaprogramming, for the functional semantic interpretation of textual sentences [22] or in planning artificial intelligence, with the help of lambda-calculus (and unification) or using combinatory logic if we want to avoid a telescoping of variables [7,11]. The interpretation of a processing chain will constitute the outcome of its underlying primitive operations and the way that these operations are organized accordingly to the principle of compositionality. The set of composed processing chains becomes a set of theorems for the proposed formal system. The system in itself is inferential. It proceeds by successive reductions of applicative categories assigned to operations concerned by the composition.

Let us now consider the applicative categorial rules of our model [5,6]:

Application rule	[X : x] + [M1 : Fxy] ---------- [(M1 X) : y]
Composition rule	[M1 : Fxy] + [M2 : Fyz] (a) ----------**B** [(**B** M2 M1) : Fxz]
	[M1 : FxFty] + [M2 : Fyz] (b) ----------$\mathbf{B^2}$ [($\mathbf{B^2}$ M2 M1) : FxFtz]
Distributive composition rule	[M1 : Fxy] + [M2 : FxFyz] ----------**S** [(**S** M2 M1) : Fxz]
Permutation rule	[M1 : FxFyz] (a) ----------**C** [(**C** M1) : FyFxz]
	[M1 : FxFyFtz] (b) ----------**C#** [(**C** ($\mathbf{C_2}$ M1)) : FtFxFyz]
Duplication rule	[M1 : FxFxy] ----------**W** [(**W** M1) : Fxy]

The premises in each applicative categorial rule are typed "connected modules", and the results are typed applicative expressions (of modules) with an eventual introduction of one combinator. These applicative expressions allow the interpretation of the processing chains. Types of modules in the premises will allow us to validate the application of the rules, and therefore to accept or reject the connection of the modules. In other words, an inferential calculation on types will allow verifying the syntactic correctness of processing chains.

Combinatory logic fills two major goals:

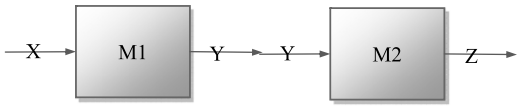

Fig. 3 – Accepted linear connection of two modules.

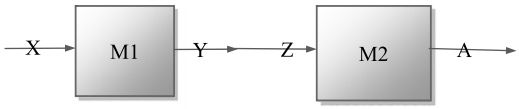

Fig. 4 – Rejected linear connection of two modules.

(i) It gives an interoperable and formal representation of the solution.

(ii) It gives the direct execution order of the modules, which form the processing chain.

Within this formal system, in order to build a processing chain, specific data is needed:

(i) The list of the modules.

(ii) The list of their inputs and outputs.

As for ACCG, other rules can be added to the formal system to improve its expressive power. The same principles as those for combinatory categorial rules characterize the set of all possible applicative categorial rules:

(1) The principle of Adjacency: Applicative categorical rules may only apply to adjacent modules. For instance, we consider modules M1 and M2 in Fig. 3 as adjacent. We also consider that M1 and M3 in Fig. 6(a) as adjacent. However, in Fig. 6(a) M2 and M3 are not adjacent.
(2) The principle of Consistency: Applicative categorical rules must be consistent with how the principal operator should apply to its operand. For instance, the principle of consistency excludes the following rule, in which M1 is the principal operator which, according to the type Fxy, must be preceded by its input with type x.

```
[M1 : Fxy] + [X : x]
--------------------------
[(M1 X) : y]
```

(3) The principle of Inheritance: This principle is implicit, since we only have one constructive operator F.

5. The implementation of the formal model

In this section we will show how our formal system works. On one hand, we will demonstrate how the applicative categorial rules given in this paper work. On the other hand, we will provide a complete analysis of a processing chain.

Let us consider first linear connection of two (or more) modules (Figs. 3–5).

The first module M1 is of type Fxy. M1 applies on the input X of type x in order to yield the output Y of type y. The second module M2 is of type Fyz. M2 applies on the input Y of type y in order to yield the output Z of type z. The graphical notation in Fig. 3 will be expressed by the following expression: [M1 : Fxy] + [M2 : Fyz]. The first composition rule, given above, returns to the complex module (**B** M2 M1) of type Fxz. In other words, the composition of M2 and M1 is possible, and the new module applies on one input of type x in order to yield one output of type y.

In the case of the example given in Fig. 4, graphical notation will be expressed by the following expression: [M1 : Fxy] + [M2 : Fza]. The first composition rule previously described does not allow the composition of M2 and M1 since the type z of the input of M2 given in the type Fza is not similar to the type y of the output of M1 given in the type Fxy. The connection of M1 and M2 is rejected.

Examples given in Figs. 3 and 4 concern the connection of two modules. In Fig. 5, we give an example of the connection of three modules (Fig. 5).

The analysis begins by connecting modules M2 (with the type is Fyz) and M1 (with the type is Fxy). By using the first composition rule, the analysis yields the complex module (**B** M2 M1) whose type is Fxz. This module is then composed with M3. The resulting module is: (**B** M3 (**B** M2 M1)) whose type is Fxa (in other words the input of the obtained complex module in this case must be of type x whereas the output must be of type a).

Overall, when we have several modules connected in a linear chain processing, analysis iterates the application of the first composition rule to modules from left to right.

The composition rule can also be used in the case of the example presented in Fig. 6 in which two modules are connected to a third one. M1 is of type Fxy. M2 is of type Fza. M3 is of type FyFau. Since M1 is of type Fxy and M3 of type FyFau, the first composition rule allows the construction of the complex module (**B** M3 M1) with the type FxFau. The processing chain given in Fig. 6(a) will thus be equivalent to the processing chain given in Fig. 6(b). In fact, the processing chain in Fig. 6(b) corresponds to the following applicative expression: (**B** M3 M1) X (M2 Z), in which X is the first operand of the complex module (**B** M3 M1), and (M2 Z) the second. However, we need to have all inputs at the most right of our expressions. To do this, we apply the first permutation rule on the category [(**B** M3 M1) : FxFau]. It yields the equivalent category [(**C** (**B** M3 M1)) : FaFxu] that corresponds to the processing chain in Fig. 6(c). We then carry on with the use of the first composition rule, given the types Fza for M2 and FaFxu for (**C** (**B** M3 M1)). We finally obtain the complex module (**B** (**C** (**B** M3 M1)) M2) whose type is FzFxu (Fig. 6(d)).

The case of Fig. 7(a) is frequently encountered in the domain of text mining. The same input is required for one or more modules, whose outputs are used as inputs for another module. This processing chain is similar to the one given in Fig. 6(a), even if the inputs of M1 and M2 are the same. The analysis will also be identical to the previous one applied to the processing chain described by Fig. 6(a). The obtained complex module will be (B (C (B M3 M1)) M2) with the type FxFxu (Fig. 7(b)). For practical reasons, we must eliminate the duplication of input X. To do this, the application of the duplication rule to the category [(B (C (B M3 M1)) M2) : FxFxu] provides a module (W (B (C (B M3 M1)) M2)) that requires a single input (Fig. 7(c)) from a module that requires two inputs. The type of the new complex module is Fxu.

We have presented some cases of basic processing chains and showed how to apply the rules of the formal model to

Fig. 5 – Accepted linear connection of three modules.

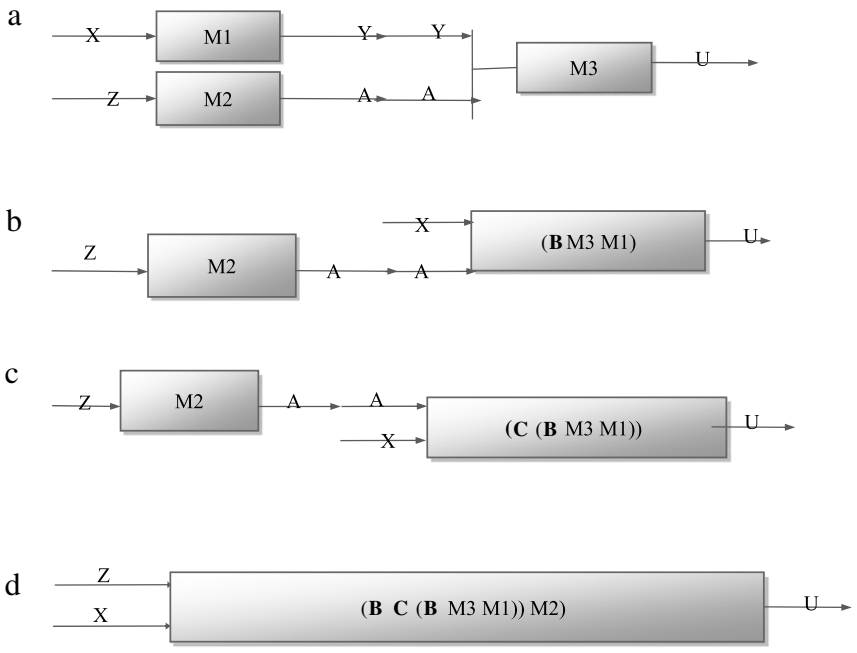

Fig. 6 – Two modules connected to a third module.

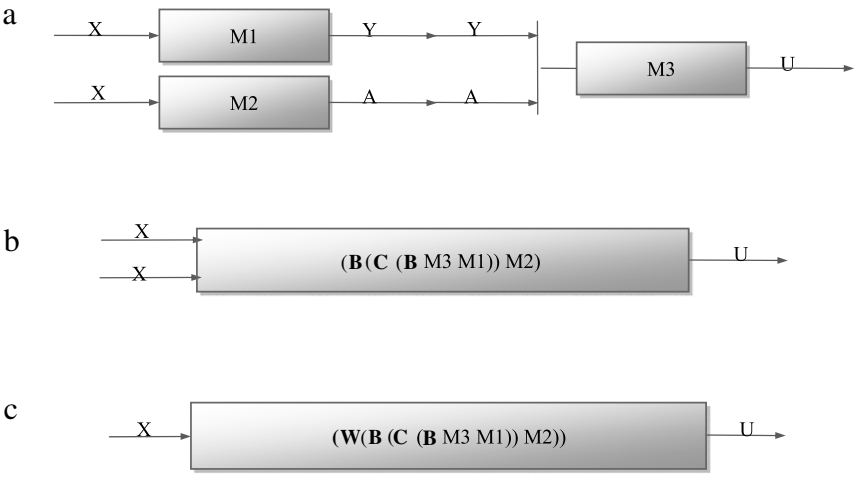

Fig. 7 – Two modules with the same input connected to a third module.

verify the syntactic correctness and to construct their applicative representation, which will allow their interpretation and their execution. All cases, we have shown, have either arrangements of modules in series or parallel arrangements. A serial processing chain is composed of many modules connected together. When a processing chain contains at least one module with more than one input, we call it a parallel processing chain.

Let us, now give an analysis based on a typed combinatory approach of a somewhat complex processing chain can look like. The processing chain given in Fig. 8 is a combination of nine modules. M1 is of type Fx_1y_1. M2 is of type Fx_2y_3. M3 is

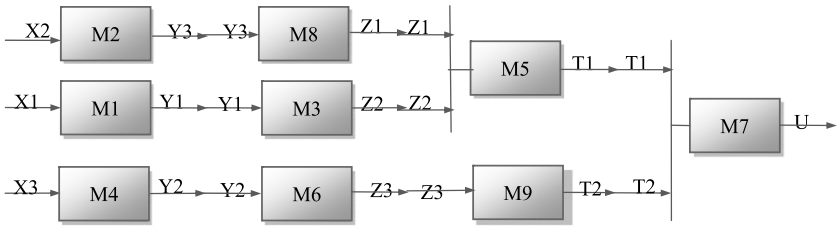

Fig. 8 – A complex processing chain.

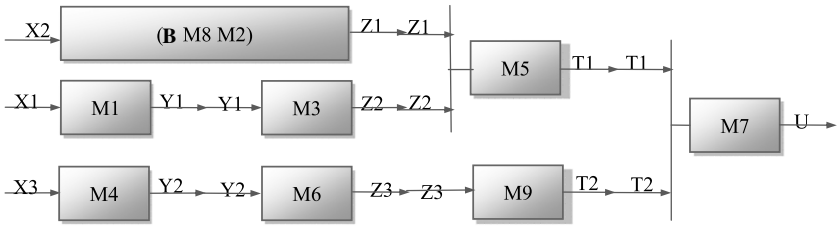

Fig. 9 – First step of the analysis.

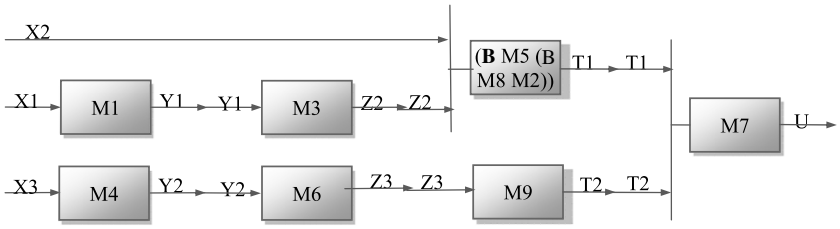

Fig. 10 – Second step of the analysis.

of type Fy_1z_2. M4 is of type Fx_3y_2. M5 is of type $Fz_1Fz_2t_1$. M6 is of type Fy_2z_3. M7 is of type Ft_1Ft_2u. M8 is of type Fy_2z1. M9 is of type Fz_3t_2.

The first step is to combine M8 and M2. Since M8 and M2 are respectively of type Fy_3z_1 and Fx_2y_3, the first composition rule is applied and the complex module (**B** M8 M2) is constructed. Its type is Fx_2z_1. The processing chain in Fig. 8 is reduced to the one in Fig. 9.

Since M5 and (**B** M8 M2) are considered adjacent, and are respectively of type $Fz_1Fz_2t_1$ and Fx_2z_1, the first composition rule is applied and complex module (**B** M5 (**B** M8 M2)) is constructed. Its type is $Fx_2Fz_2t_1$. The processing chain in Fig. 9 is reduced to the one in Fig. 10.

This step consists in combining M3 and M1. Since M3 and M1 are respectively of type Fy_1z_2 and Fx_1y_1, the first composition rule is applied and the complex module (**B** M3 M1) is constructed. Its type is Fx_1z_2. The processing chain in Fig. 10 is reduced to the one in Fig. 11.

At this step, complex modules (**B** M3 M1) and (**B** M5 (**B** M8 M2)) are not considered as adjacent, we must apply the first permutation rule on the category (**B** M5 (**B** M8 M2)). We, then, construct the complex module (**C** (**B** M5 (**B** M8 M2))) whose type is $Fz_2Fx_2t_1$ (see Fig. 12).

Complex modules (**B** M3 M1) and (**C** (**B** M5 (**B** M8 M2))) are now adjacent and of respective types Fx_1z_2 and $Fz_2Fx_2t_1$. We must then apply the first composition rule. The complex module (**B** (**C** (**B** M5 (**B** M8 M2))) (**B** M3 M1)) is constructed. Its type is $Fx_1Fx_2t_1$ (see Fig. 13).

Modules M7 and (**B** (**C** (**B** M5 (**B** M8 M2))) (**B** M3 M1)) are adjacent. Since their respective types are Ft_1ft_2u and $Fx_1Fx_2t_1$, we must apply the second composition rule. The complex module ($\mathbf{B}^2$ M7 (**B** (**C** (**B** M5 (**B** M8 M2))) (**B** M3 M1))) is constructed. Its type is $Fx_1Fx_2Ft_2u$ (see Fig. 14).

We, now, have to combine M4, M6 and M9. We will not show the details of this combination. Simply refer to the analysis above of the processing chain given in Fig. 5. The analysis yields the complex module (**B** M9 (**B** M6 M4)) whose type is Fx_3t_2 (see Fig. 15).

At the eighth step (Fig. 16) the analysis applies the second permutation rule to ($\mathbf{B}^2$ M7 (**B** (**C** (**B** M5 (**B** M8 M2))) (**B** M3 M1))). This operation yields the complex module (**C** ($\mathbf{C}_2$ ($\mathbf{B}^2$ M7 (**B** (**C** (**B** M5 (**B** M8 M2))) (**B** M3 M1))))) whose type is $Ft_2Fx_1Fx_2u$ (Fig. 16). The last step concerns the application of the first composition rule to complex modules (**C** ($\mathbf{C}_2$ ($\mathbf{B}^2$ M7 (**B** (**C** (**B** M5 (**B** M8 M2))) (**B** M3 M1))))) and (**B** M9 (**B** M6 M4)) since their respective types are $Ft_2Fx_1Fx_2u$ and Fx_3t2. It yields the complex module (**B** (**C** ($\mathbf{C}_2$ ($\mathbf{B}^2$ M7 (**B** (**C** (**B** M5 (**B** M8 M2))) (**B** M3 M1))))) (**B** M9 (**B** M6 M4))) whose type is $Fx_3Fx_1Fx_2u$ (Fig. 17). At this last step the processing chain is considered to be syntactically correct and the complex module expressed by means of combinators is the combinatory expression underlies this processing chain. The reduction of combinators

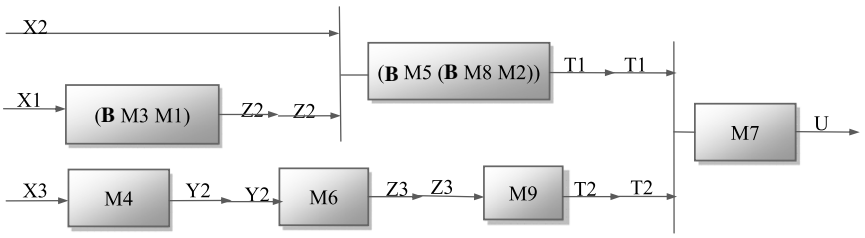

Fig. 11 – Third step of the analysis.

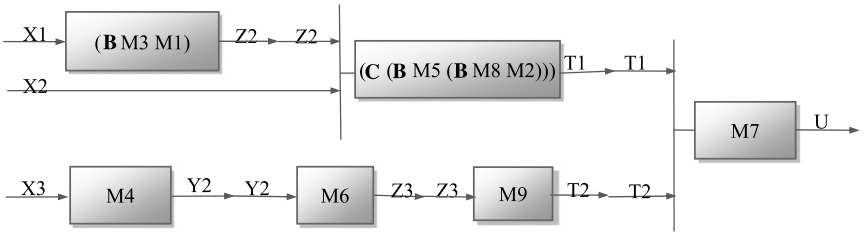

Fig. 12 – Fourth step of the analysis.

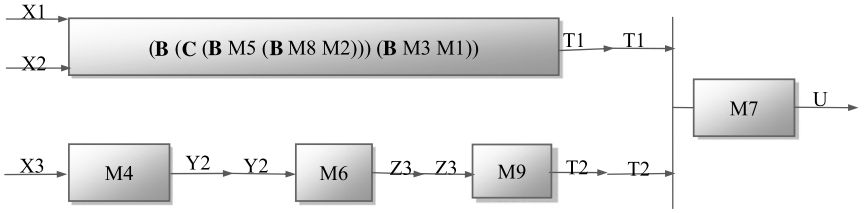

Fig. 13 – Fifth step of the analysis.

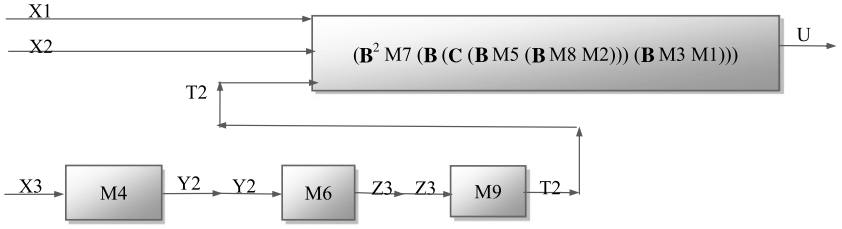

Fig. 14 – Sixth step of the analysis.

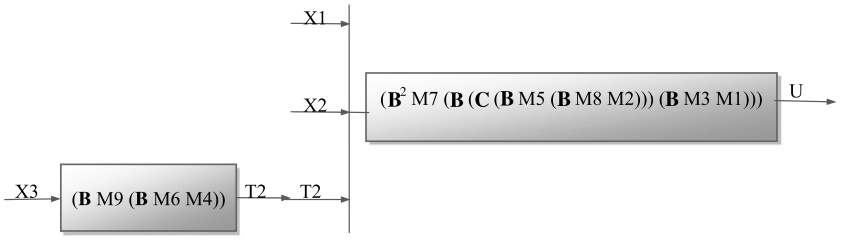

Fig. 15 – Seventh step of the analysis.

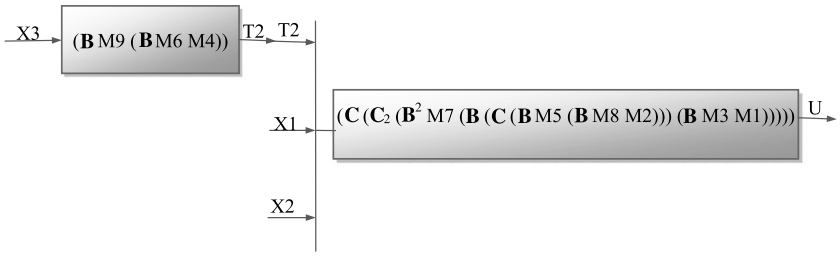

Fig. 16 – Eighth step of the analysis.

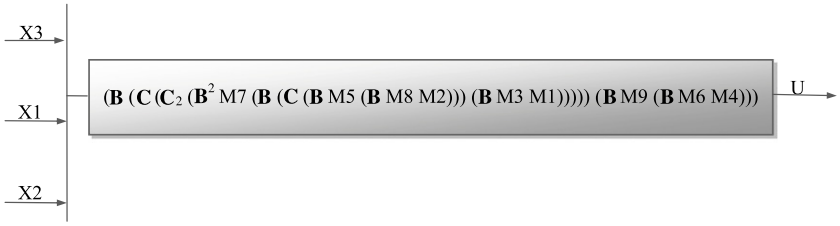

Fig. 17 – Last step of the analysis.

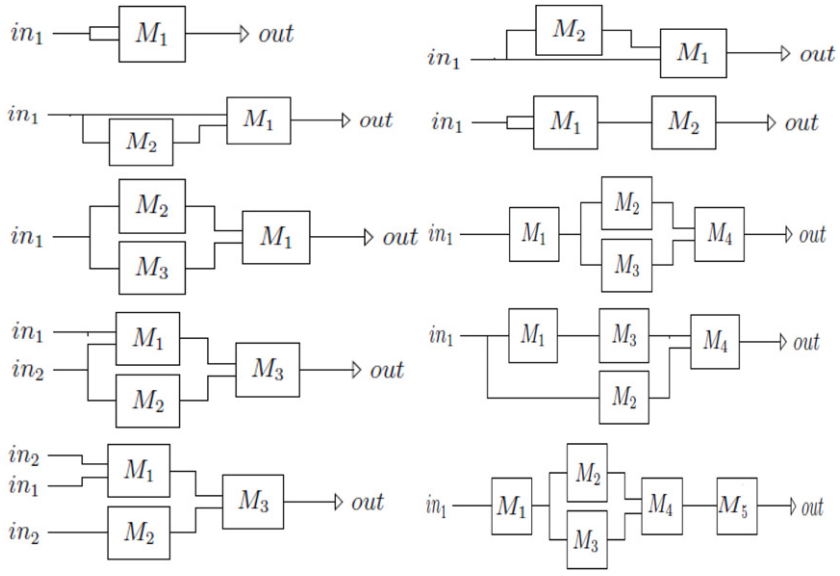

Fig. 18 – Complex processing chains.

from left to right by means of β-reduction rules gives the order in which each module can be running on its inputs (in this case inputs are X1, X2 and X3).

We show below steps of the reduction of the combinatory expression (**B** (**C** (**C**$_2$ (**B**^{2} M7 (**B** (**C** (**B** M5 (**B** M8 M2))) (**B** M3 M1))))) (**B** M9 (**B** M6 M4))).

On the right column, we specify what combinator has been reduced (see Box I).

A prototype of the theoretical model was implemented in C#. We tested several particular arrangements of serials, parallels and output-distributed modules as well as complex processing chains as illustrated in Fig. 18.

6. Conclusion

The need for flexible, adaptable, consistent and easy-to-use tools and platforms, in a recent and active field, such as information retrieval, categorization, classification, indexation, language processing, knowledge processing, etc., is essential. Tasks from these domains are complex, very demanding and many challenges are yet to be solved; when it comes to take into account (i) the point of view and the perspective of the user; (ii) the emergence of new needs that require new methods and new processing chains.

The development of new methods and new processing chains demands thoughtful approaches, analysis, modelization, implementation, and testing. Indeed, a processing chain

(B (C (C_2 (B^2 M7 (B (C (B M5 (B M8 M2))) (B M3 M1))))) (B M9 (B M6 M4))) X3 X1 X2	
(C (C_2 (B^2 M7 (B (C (B M5 (B M8 M2))) (B M3 M1))))) ((B M9 (B M6 M4)) X3) X1 X2	B
(C_2 (B^2 M7 (B (C (B M5 (B M8 M2))) (B M3 M1)))) X1 ((B M9 (B M6 M4)) X3) X2	C
(B^2 M7 (B (C (B M5 (B M8 M2))) (B M3 M1))) X1 X2((B M9 (B M6 M4)) X3)	C_2
M7 ((B (C (B M5 (B M8 M2))) (B M3 M1)) X1 X2) ((B M9 (B M6 M4)) X3)	B^2
M7 ((B M5 (B M8 M2)) X2((B M3 M1) X1))((B M9 (B M6 M4)) X3)	C
M7 (M5 ((B M8 M2) X2) ((B M3 M1) X1)) ((B M9(B M6 M4)) X3)	B
M7 (M5 (M8 (M2 X2)) ((B M3 M1) X1)) ((B M9 (B M6 M4)) X3)	B
M7 (M5 (M8 (M2 X2)) (M3 (M1 X1))) ((B M9 (B M6 M4)) X3)	B
M7 (M5 (M8 (M2 X2)) (M3 (M1 X1))) (M9 ((B M6 M4) X3))	B
M7 (M5 (M8 (M2 X2)) (M3 (M1 X1))) (M9 (M6 (M4 X3)))	B

Box I.

must be the result of a true discovery process that requires constant back and forth between theoretical description of the solution, software implementation, testing and refinement of the theoretical description in the light of the results of experimentation. This process is iterative.

Some projects based on this philosophy have seen the light in the last years. The model we suggest has strong formal foundations (Applicative and Categorial Grammars, and combinatory logic). It allows rapid prototyping and supports a maximal re-use and composition of existing modules. One of the principal advantages of this formalism is to ensure a firm compositionality of the different modules in the different processing chains.

Text analysis is only a modality of the theoretical framework developed here. It is possible to adapt this work to other types of data. A perspective that gives us the privilege to witness interesting developments and relevant collaborations with various disciplines like software engineering, big data, cloud computing and Internet of things.

Acknowledgment

This work was supported by a grant from Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] H. Anoun, Towards a logical approach to nominal sentences analysis in standard arabic. in: Proceedings of ESSLI'06, 2006.

[2] J. Baldridge, G.J. Kruijff, Multi-modal combinatory categorial grammar. in: Proceedings of EACL'03, 2003.

[3] M. Bertin, I. Atanassova, Extraction and characterization of citations in scientific papers, in: Valentina Presutti, Milan Stankovic, Erik Cambria, Iván Cantador, Angelo Di Iorio, Tommaso Di Noia, Christoph Lange, Diego Reforgiato Recupero, Anna Tordai. (Eds.), Semantic Web Evaluation Challenge, Springer Publishing, 2014.

[4] M. Bertin, I. Atanassova, J.P. Desclés, Extraction of authors' definitions using indexed reference identification. in: Proceedings of Recent Advances in Natural Language Processing, Bulgaria, 2009.

[5] I. Biskri, B. Amar Bensaber, A flexible approach for text processing Engineering. in: Proceedings of ACM-MEDES'14, Buraidah, 2014.

[6] I. Biskri, M. Anastacio, A. Joly, B. Amar Bensaber, Integration of sequence of computational modules dedicated to text analysis: a combinatory typed approach. in: Proceedings of AAAI-FLAIRS'13, St-Pete Beach, 2013.

[7] I. Biskri, Applications linguistiques multilingues destinées au WEB: Apport des Grammaires Catégorielles, in: Jean Pierre Desclés, Florence Le Priol. (Eds.), Dans Annotations automatiques et recherche d'informations, Presses Hermès, Paris, 2009.

[8] C. Bozsahin, The combinatory morphemic lexicon, Comput. Linguist. 28 (2) (2012) 145–186.

[9] G. Crispino, S. Ben Hazez, J.L. Minel, Architecture logicielle de Context; plate-forme d'ingénierie linguistique, in: Proceedings of TALN'99, 1999.

[10] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, GATE: A framework and graphical development environment for robust nlp tools and applications. in: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics, ACL'02, Philadelphia, 2002.

[11] B.H. Curry, R. Feys, Combinatory Logic, Vol. I, North-Holland, 1958.

[12] P. De Groote, S. Podogalla, On the expressive power of Abstract Categorial Grammars: Representing context-free formalisms, J. Logic Inform. (2004).

[13] J.P. Descles, Langages Applicatifs, Langues Naturelles et cognition, Hermes, Paris, 1990.

[14] J.S. Downie, J. Unsworth, B. Yu, D. Tcheng, G. Rockwell, S.J. Ramsay, A revolutionary approach to humanities computing: tools development and the D2K datamining framework. in: Proceedings of the 17th Joint International Conference of ACH/ALLC. 2005.

[15] J.R. Hindley, J.P. Seldin, Lambda-calculus and Combinators, an Introduction, Cambridge University Press, 2008.

[16] J. Kang, Problèmes morpho-Syntaxiques Analysés dans un Modèle catégoriel étendu: Application au coréen et au Français avec une Réalisation Informatique, Thèse de Doctorat Paris Sorbonne, 2011.

[17] Y. Kubota, A Multimodal Combinatory Categorial Grammar analysis of –te form complementation. Colloque de syntaxe et sémantique, Paris, 2007.

[18] W. McCarty, Humanities Computing, Palgrave Macmillan, Basingstoke, 2005.

[19] I. Mierswa, M. Wurst, R. Klinkemberg, M. Scholz, T. Euler, YALE: rapid prototyping for complex data mining tasks, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD, 2006), ACM Press, 2006.

[20] M. Pentus, Lambek grammars are context-free, in: In Proceedings of the IEEE Symposium on Logic in Computer Science, Montreal, 1993.

[21] S.K. Shaumyan, Two paradigms of linguistics: the semiotic versus non-semiotic paradigm, Web J. Formal, Comput. Cognitive Linguist. (1998).
[22] M. Steedman, The Syntactic Process, MIT Press/Bradford Books, 2000.
[23] A.W. Warr, Integration, analysis and collaboration. An update on workflow and pipelining in cheminformatics, Strand Life Sci. (2007).
[24] J. Unsworth, Scholarly primitives: what methods do humanities researchers have in common, and how might our tools reflect this, in: Humanities Computing: Formal Methods, Experimental Practice Symposium, King's College, London, 2000.
[25] I. Witten, E. Frank, M. Hall, Data mining: practical machine learning tools and techniques, Morgan Kaufmann Publishers, 2011.